Haaga-Helia
ammattikorkeakoulu Oy

Yrityksen varauspalvelun toteutus

Pekka Koukkula

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2020

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Pekka Koukkula

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Yrityksen varauspalvelun toteutus

Sivu- ja liitesivumäärä
28 + 2

Tämän toiminnallisen opinnäytetyön aiheena on Yrityksen varauspalvelun toteutus. Sovelluksella pyritään kehittämään toimeksiantajan palvelua asiakkaille ja yrityksen liiketoimintaa. Sovellus tehdään osaksi yrityksen muuta liiketoimintaa ja tulee olemaan olennainen osa sen digitaalista palvelua.

Toimeksiantona suunnitellaan ja kehitetään Kiinteistö Oy Pyynpesä:lle sovellus, jonka avulla yrityksen asiakkaat pystyvät varaamaan yrityksen tarjoamista varastopalveluista itselleen sopivan varaston. Yrityksen asiakkaat pystyvät varaamisen lisäksi hoitamaan maksamisen sovelluksen kautta ja hallitsemaan omia henkilö- ja tilaustietojaan. Tämän ansiosta yrityksen manuaalinen työn määrä vähenee ja digitaalisen palvelun käyttö onnistuu. Yrityksellä ei aikaisemmin vastaavaa sovellusta ole.

Opinnäytetyö tehdään vetoketjumallilla, eli luvuissa empiirinen osio esitetään tietoperusteisen osion jälkeen. Tämän opinnäytetyön luvuissa käsitellään sovelluksen työkaluja ja menetelmiä, rakennetta ja työn vaiheita sekä tietoturvallisuuden käsittelyä. Työstä rajataan pois testaus sekä sovelluskehityksen palvelinpuolen kehitys, kuten rajapinta sekä tietokanta. Työssä keskitytään sovelluksen suunnitteluun ja teknisen puolen toteutustapoihin. Tietoperusteisissa osioissa avataan termejä sekä pohjustetaan valittuja menetelmiä että teknologioita. Työn tekeminen aloitetaan helmikuussa 2020 ja lopetetaan kesäkuussa 2020. Työn alkuvaiheessa toteutetaan sovelluksen arkkitehtuurinen suunnitelma ja käyttöliittymäsuunnittelu.

Lopputuloksena syntyy digitaalinen sovellus, jonka kehitys jatkuu vielä tämän toimeksiannon jälkeen. Sovelluksen avulla yritys voi kasvattaa liiketoimintaansa ja sovelluksen kehittäminen uusiin toimintoihin on jatkossa helpompaa. Jatkokehitykseen kuuluu ainakin maksuominaisuuksien monipuolistaminen ja eri paikkakuntien lisäys yrityksen toiminnan kasvaessa.

Asiasanat
Ohjelmistokehitys, ohjelmointi, Angular, Front-End, ohjelmistosuunnittelu, tietoturva

Sisällys

1 Johdanto

Tänä päivänä jopa yksityishenkilöille kertyy usein tavaroita, joita he haluavat säilöä tai varastoida jotenkin, mutta tila yksityisasunnoissa on valitettavan usein rajallinen. Varsinkin suurissa asuinkeskuksissa kaupungeissa yksityishenkilöillä on valitettavan vähän tilaa hankkia tavaroita. Tutkimuksen mukaan 54 prosenttia 35-54-vuotiaista suomalaisista kokee, että kodeissa on liian vähän tilaa tavaroille. Pienvarastojen vuokraus liiketoimintana on yleistynyt selvästi 2000-luvulla. Kasvua selittää kasvukeskusten asuntojen pienuus ja varastotilan rajallisuus. Ihmiset miettivät, kannattaako isoa asuntoa ostaa vain tavaroiden takia. (Suomen kiinteistölehti 2017; Kempas 2017.)

Samaan aikaan meneillään on mullistus yritysten kulisseissa, miten erilaiset palvelut tuotetaan digitaalisesti (Telia 2016). Menestyvien yritysten liiketoiminnan ytimessä on digitaalisuus. Kun yritykset kuitenkin kertovat kehittävänsä digitaalista liiketoimintaa, se tarkoittaa jo olemassa olevan liiketoiminnan oheen rakennettavaa digitaalista asiaa. Tämä voi usein tarkoittaa käyttäjille näkymättömissä tapahtuvaa prosessien automatisointia, joka tuo liiketoiminnalle tehokuutta ja kilpailukykyä. (Digia 2019, 3-4.)

Tämän toiminnallisen opinnäytetyön tavoitteena oli sovellus, joka on yrityksen varastojen varauspalvelusovellus. Toimeksiannon tarpeena oli saada aikaan ohjelmisto, jonka avulla yrityksen asiakkaat voivat tilata itselleen yrityksen varastopalvelun. Sovelluksen tavoitteena oli houkutella asiakkaita käyttämään yrityksen palveluita. Yrityksen asiakkaat voivat hallinnoida jatkuvia tilauksiaan sovelluksen avulla sekä keskeyttää tilauksiaan ilman yhtään yhteydenottoa asiakaspalveluun. Tämän ansiosta asiakkaiden tiedot rekisteröitiin automaattisesti järjestelmään, jonka avulla he voivat kulkea varastorakennukseen ja hallita omaa varastoaan. Tämä samainen sovellus suoraviivaisti palvelun käyttöä, joka osaltaan tehosti toimeksiantajan liiketoimintaa. Se mahdollisti yrityksen toimintaan kilpailuetua muihin alueen vastaaviin toimijoihin verraten ja tekee liikeideasta paremmin skaalattavan.

Opinnäytetyön keskeisiä asioita olivat sovelluksen kehittäminen ja sovelluksen tietoturva. Rajasin työstä pois sovelluksen testaamisen sekä sen ympärille rakennettavat muut tarvittavat toiminnot, kuten rajapinnan sekä tietokannan. Rajasin työn aiheet tietoturvallisuuden yleisiin käsitteisiin ja tietoturvan saavuttamiseksi käytettyihin menetelmiin sekä sovelluksen kehittämisessä käytettyihin menetelmiin ja miksi näin tehtiin. Tietoturvallisuus ja sovelluksen arkkitehtuuriset ratkaisut olivat hyvinkin tärkeässä osassa, sillä sovellusta ei voitu ottaa käyttöön, jos se ei täytä tietoturvan vähimmäiskriteerejä.

Digitaalisten palveluiden hyödyt voidaan jakaa kolmeen osioon: myynnin kasvattaminen, asiakastyytyväisyys ja kustannustehokkuus. Olennaista digitaalisten palveluiden kehittämisessä on tarkastella asioita asiakkaan näkökulmasta. Keitä asiakkaat ovat ja miten he haluavat asioida. Digitaalisten palveluiden kehittämisen keskiössä on asiakas. Saumaton digitaalinen kokonaisuus antaa positiivisen ja miellyttävän käyttökokemuksen asiakkaalle. Eri digitaalisten palveluiden integraatiot ovat kiinnostavia ja entistä enemmän liiketoiminnan kiinnostuksen kohteita. (Digia 2019, 12-13.)

Tämän sovelluksena tehtävän työn tavoitteena oli luoda palvelun virtaviivaistus, jonka avulla asiakas voi kokea palvelun vaivattomaksi ja helpoksi. Toimeksiantajan yritys voi tämän ansiosta keskittää resurssejaan muihin tehtäviin kuin manuaalisiin toimintoihin. Sovellus tuotettiin jo olemassa olevan kotisivun ja järjestelmän ohelle ja sen käyttöön asiakkaat pääsivät suoraan kotisivujen kautta. Sovellus yhdistettiin yrityksen olemassa olevaan pilvipalvelimella toimivaan rajapintaan, jonka kautta sovelluksella asiakas voi rekisteröityä tai kirjautua omiin tietoihinsa.

Opinnäytetyön toimeksiantajana oli Torniossa toimiva kiinteistöyritys Kiinteistö Oy Pyynpesä, jonka tavoitteena oli digitalisoida toimintojaan. Yritys vuokraa kiinteistöjä, mutta uutena toimialana vuokraa pienvarastopalvelua yksityisille sekä yrityksille. Uuden liiketoiminnan tavoitteena on mahdollistaa asiakkaille helppo oman varaston vuokraaminen ja kulku omaan vuokraamaansa varastoon aivan Tornion ydinkeskustassa.

Yritys on muutaman hengen mikroyritys, mutta on toiminut jo useamman vuoden ajan kiinteistöalalla. Yrityksen asiakkaat pystyvät kulkemaan sen varastotiloihin omalla rekisteröidyllä puhelinnumerollaan, joten sillä tavalla kyettiin luomaan turvallisuutta varaston käyttäjille, kun tiedettiin, kuka oli kulkenut varastoon ja milloin.

Tämä raportti toteutettiin vetoketjumallilla, koska avattavia termejä ja konsepteja voi olla ulkopuoliselle useampia. Pyrin avaamaan termejä kappaleiden alussa ja välissä, jotta lukija pystyy ymmärtämään kokonaisuutta.

2 Työn rajaus, työssä käytetyt työkalut ja menetelmät

Sovelluksen kehittäminen on yhdistelmä luomista, suunnittelua, julkaisua ja ohjelman ylläpitoa. Ohjelma itsessään on vain kokoelma ohjeita, joka kertoo mitä tietokone tekee. Sovellukset auttavat käyttäjiä suorittamaan jonkin tehtävän, johon käyttäjä tarvitsee ohjelmaa. Ohjelmistokehitys on tapa yrityksille erottautua joukosta ja olla kilpailullisempi. Sillä voidaan kehittää tapoja muokata jokaista palvelua erilaiseksi ja kehittää jopa perinteisiä liiketoimintoja. (IBM 2020.)

Tietoturvalla tarkoitetaan toimia, joilla varmistetaan tiedon luottamuksellisuus, eheys ja käytettävyys. Tietojen täytyy olla vain niiden käyttöön oikeutettujen saatavilla, eikä niitä voi muuttaa kuin siihen oikeutetut ja tiedot sekä tietojärjestelmät ovat niiden käyttöön oikeutettujen hyödynnettävissä. (Kyberturvallisuuskeskus 2020.)

2.1 Toimeksiannon tarve

Digitalisaatio koskettaa kaikkia yrityksiä, sillä se koskettaa yritysten asiakkaita. Digitalisaation vaikutus on vuodesta 2016 muuttunut valtavasti. Kuluttajat etsivät tietoa ja tekevät hankintoja verkossa. Yritysten kilpailijat hyödyntävät varmasti erilaisia digitalisaation mahdollisuuksia. Olemassa olevan liiketoiminnan digitalisointi onkin monille yrityksille mahdollisuus. Se voi tarkoittaa käyttäjille näkymättömissä olevan prosessin automatisointia, mikä tuo liiketoiminnalle tehokkuutta ja kilpailukykyä. Nykypäivän ihmisillä on enemmän vaihtoehtoja kuin koskaan ja niinpä asiakaskokemuksen merkitys menestystekijänä korostuu. (Digia 2019, 6-7.)

Opinnäytetyön tarve ilmeni yrityksen tarpeesta digitalisoida tuleva liiketoiminta, jossa sekä fyysiset toiminnot että digitaaliset yhdistyvät automaattiseksi kokonaisuudeksi. Tämän opinnäytetyön toimeksiantajan tämänhetkinen toiminta tapahtui pääasiassa sähköpostilla tai puhelimella soittamalla asiakaspalveluun. Tämän jälkeen maksutiedot lähetettiin ja maksutapahtuman jälkeen se aktivoi varaston ulko-ovessa olevan lukon, joka toimi asiakkaan puhelinnumerolla. Yrityksen palvelu muodostui tuolla hetkellä useista manuaalisista toiminnoista, joita tällä palvelulla pyrittiin vähentämään. Näiden seikkojen vuoksi digitaalinen palvelusovellus auttoi yritystä erottautumisessa sen paikallisista kilpailijoista.

2.2 Työssä käytetyt työkalut

Integroidut ohjelmankehitysympäristöt eli IDE:t ovat ohjelmien kehittämiseen tarkoitettuja sovelluksia, jotka yhdistävät yleiset kehittäjän työkalut yhteen graafiseen käyttöliittymään.

Sen työkalut automatisoivat toistoa vaativia tehtäviä ja avustaa kehittäjää pääsemään tekemään sovelluksia ilman manuaalisia säätöjä tai asennuksia. Sen ominaisuudet on suunniteltu säästämään aikaa ja nopeuttamaan tiimien yhtenäistä työskentelyä. (Red Hat s.a.)

Visual Studio Code (lyhyemmin VS Code) on monien käyttäjien hyödyntämä ohjelma koodaamiseen sen keveyden, monipuolisuuden ja avoimen lähdekoodin vuoksi. VS Coden editori antaa käyttäjälle hyvin vapaat kädet lisätä haluamia lisäosia helpottaakseen työn tekemistä. Vaikka VS Code on monipuolinen, se ei ole integroitu ohjelmankehitysympäristö ja näin ollen vaatii manuaalista asennusta. Se ei ole täydellinen, mutta sen avoimen lähdekoodin ansiosta jokainen käyttäjä voi halutessaan tehdä omia lisäosia siihen tai oman tarvittavan lisäosansa. Se on myös sen hyödyllisin ominaisuus. (Bolton 2017.)

Käytin työn tekemiseen Microsoftin ilmaista Visual Studio Code -koodausohjelmaa, joka oli ilmainen ja asennettavissa Windowsille, Linuxille tai Applen Maceille. Käytin työssäni kyseistä ohjelmaa ohjelmistokehitykseen, joten se oli minulle tuttu käyttää. Sen monipuolinen valikoima sisäisiä työkaluja auttoi minua kehittämään työtä nopeammin ja keskittymään tekemään vain tiettyjä asioita. Tämän lisäksi Visual Studio Codeen on helppo lisätä kiinnostavia lisäosia, joita saa ohjelman sisällä asennettua ilmaiseksi. En kuitenkaan tarvinnut työssäni mitään lisäosia, vaikka niiden käyttö olisi ollut mahdollista.

Adobe XD on markkinoilla varsin uusi ja suuren yrityksen viimeisimpiä tuotteita sen laajassa ja arvostetussa tuoteperheessä. Adobe XD kuuluu Adoben Creative Cloud jäsenyyteen, mutta sitä voi käyttää myös ilmaiseksi rajoitetuin ehdoin. Sen käyttöliittymä on hyvin samankaltainen muiden Adoben ohjelmien kanssa, joten jos tuotteet ovat tuttuja, XD:n käytön oppii todella nopeasti. Sen kevyt ja nopea käyttöliittymä hoitaa itsestään paljon taustalla tapahtuvia raskaita toimenpiteitä sekä siihen on mahdollista hankkia paljon erilaisia lisäosia suunnittelua edistämään. Adobe itse jopa kannustaa lisäosien tekoon ja käyttämiseen. (Valishvili 2018.)

Käyttöliittymäsuunnitteluun käytin Adoben XD -ohjelmaa, joka oli tarkoitettu käyttöliittymäsuunnitteluun. Adobe XD:llä oli ollut vaivatonta tehdä suunnitelmallisia pohjia. Kyseisessä ohjelmassa on rajoitetulla toiminnallisuudessa tehtävä testausmoodi, jossa eri painikkeille voi tehdä simulointia tulevan ohjelman toiminnasta. Tämä helpotti toiminnallisuuden demonstrointia toimeksiantajalle. XD:ssä alussa valitsin pohjaksi verkkosivun yleisen koon valmiiksi asetetuista pohjista ja työstin yleiset näkymät kerrallaan. Ohjelmassa pystyi luomaan erilaisia muotoja ja tekstielementtejä vain vetämällä niitä paikoilleen. Tällä tavalla yksi elementti kerrallaan muodostin pari eri käyttöliittymävaihtoehtoa.

2.3 Työssä käytetty ohjelmistokehys Angular

Verkkosivuilla on kaksi puolta, selainpuoli (Front-End) ja palvelinpuoli (Back-End). Selainpuoli tarkoittaa kaikkea sitä koodia, mikä tapahtuu käyttäjän verkkoselaimessa ja jonka lähdekoodia voi kuka tahansa lukea omassa selaimessaan. Palvelinpuoli taas ajetaan sivuston palvelimella, esimerkiksi yrityksen serverihuoneessa tai pilvipalvelussa. Palvelinpuolella tapahtuvat salaisena pidetyt käsittelyt, kuten salasanojen tallennus, järjestelmäintegraatiot ja tietokantojen käsittely. (Laine 2015.)

Angular on Googlen kehittämä selainpuolen ohjelmistokehys, jonka ensimmäinen versio oli nimeltään AngularJs, mutta myöhemmin kehitettiin täysin uusiksi ja nimettiin pelkäksi Angular:ksi. Nimi oli alun perin Angular 2, mutta sittemmin se on kasvanut ja saanut uusia versioita, joten siitä käytetään vain pelkästään Angular. Vaikka monissa paikoissa yhä käytetään AngularJs:ää, on se aivan erilainen kuin Angular. Tämän vuoksi näitä kahta ei pidä sekoittaa keskenään. (Fain & Moiseev 2017, 1-2.)

Miksi Angular on niin suosittu? Sovelluskehitys on usein monimutkainen prosessi ja sovellusten teko nopeaa on haaste. Angular sisältää monia toimintoja jo oletuksena sisällään, joita näitä testataan ja käsitellään keskenään, jotta toimivuus voidaan taata. Jokainen voi käyttää jo sisällytettyjä työkaluja tai vaihtaa ne helposti mihin tahansa haluamaansa kirjastoon. Angular on kehitetty Googlen skaalan ongelmien ratkomiseen. Angularissa käytetty komponenttimalli jakaa tehtävät työt eri palasiin, joita on eri tiimin jäsenten helpompi tehdä. Sen sisältämä TypeScript helpottaa ongelmien löytämistä jo alkuvaiheessa. Typescript on Javascriptin päälle rakennettu avoimen lähdekoodin ohjelmointikieli, joka lisää Javascriptiin staattiset tyypit. Tyypit antavat mahdollisuuden määritellä ohjelmointiolion muodon ja tyypin. Typescript on kuitenkin täysin vapaaehtoinen ja kaikki Javascript koodit ovat Typescript koodia. (Fluin 2017; TypescriptLang 2020.)

Kehitystiimi Angularin takana tekee jatkuvasti työtä sen eteen, että sitä käyttävä yleisö tietää mitä on tulossa ja mitä muutetaan. Angular on Googlen tuote, joten se hyödyntää Googlen testausinfrastruktuuria. Tämä tarkoittaa, että kaikki muutokset pitää toimia kaikissa Googlen omissa Angular projekteissa. Tämä tekee siitä luotettavan, kun jokainen versio on hyvin testattu ennen julkaisua. Angular on helppo omaksua niille, jotka tulevat AngularJs tai Javascript taustasta. Toinen helppo tausta on Java ja C# koodaajille, koska sovellusten arkkitehtuuri on usein samankaltainen kuin heidän käyttämissään koodeissa. (Fluin 2017.)

Angular 2 on huomattavasti tehokkaampi kuin AngularJs. Angularille tyypilliseen tapaan yksi moduuli on yksi tiedosto. Moduuleiden käyttäminen toisten moduuleiden sisällä on

erittäin helppoa. Jokaisen moduuleiden sisälle voi tehdä niin monta eri komponenttia kuin haluaa. Komponentit muodostavat oman rajatun koodin ja komponenttia voi käyttää niin monta kertaa kuin haluaa. Komponentti muodostuu HTML, CSS ja Typescript tiedostoista. (Fain & Moiseev 2017, 8-10.)

HTML on lyhennelmä sanoista Hypertext Markup Language. Se on verkkosivun perusrakenne eli luuranko. Se on ensimmäinen asia, jonka verkkoselain lataa ja sisältää tekstit, kuvat, videot, painikkeet jne. CSS on lyhenne sanoista Cascading Style Sheet. Se antaa kaiken tyylittelyn HTML-tiedostolle. Javascript on koodikieli, jonka tehtävänä on antaa verkkosivulle toiminnallisuuden. Sillä on nykyään mahdollista tehdä melkein mitä vain verkkosivulla. (Backes 2019.)

Ohjelmistokehys on monien ohjelmistokehittäjien valmiiksi suunnittelema, testaama ja tuottama työkalu auttamaan muiden ohjelmistojen kehittämistä. Sen tarkoitus on helpottaa työskentelyä ohittamalla käyttäjältään ne monimutkaiset prosessit, jotka ohjelmistokehityksen taustalla on. Näin ollen sen käyttäjien ei tarvitse huolehtia kuin koodista, jonka he haluavat tehdä. Jos ohjelmistokehys on tehty avoimella lähdekoodilla, jokainen käyttäjä voi itse tehdä muutoksia omaan ohjelmaansa näin halutessaan ja tutustua, miten kyseinen ohjelmistokehys toimii. (Singh 2020.)

Angular-sovellukset ovat niin kutsuttuja yksisivuisia sovelluksia. Single Page Applikaatiot (lyhennettynä SPA), jotka ovat nopeita ja kevyitä. Yksisivuinen sovellus on sovellus, jonka logiikka on tuttua monista sähköposteista ja sosiaalisen median alustoista. Sen sijaan, että jokaisella toiminnolla tehdään uusi kutsu hakea koko sivun tiedot palvelimelta, haetaan SPA:lla kaikki toiminnallisuudet ja tiedot kerralla, mutta näytetään vain tarvittava. Tavallisesti verkkosivu lataa kaiken sisällön kerralla, mutta jos nettiyhteys hidastelee, niin silloin mitään ei tapahdu. (Halme 2018.)

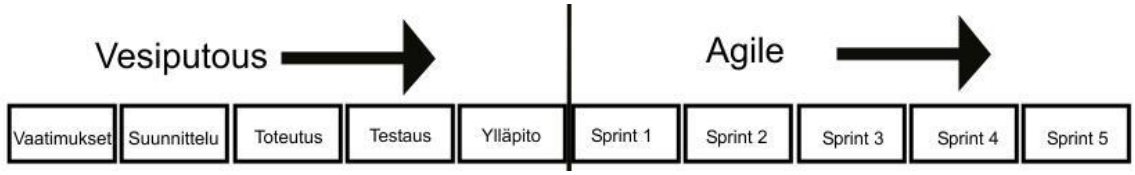
Toteutin sovelluksen Angular ohjelmistokehyksellä. Tavoitteena oli, että sovelluksesta pyritään tekemään mahdollisimman selkeä ja yksinkertainen käyttää, jotta asiakkaiden olisi jatkossakin helppo hankkia palvelunsa sitä kautta. Sovelluksesta oli tarkoitus kehittää kasvatettava, jotta asiakasyrityksen kasvaessa siihen voisi helposti lisätä erilaisia lisätoimintoja. Opinnäytetyön aiheena oleva sovelluksen tuottaminen oli sinänsä yksinkertainen konsepti, mutta tekninen toteutus vaati huomattavasti tietotaitoa, jotta sovellus voidaan ottaa tietoturvallisesti käyttöön.

Tämän sovelluksen kehittämiseen Angular sopi oikein mainiosti sen modulaarisuuden vuoksi. Halusin alusta alkaen jakaa sovelluksen eri toiminnot erilaisiin komponentteihin,

jotta pystyin testata ja rakentaa jokaista komponenttia erillään muista komponenteista. Sen ansiosta pystyin keskittymään yhteen palaan isosta kokonaisuudesta kerrallaan. Jatkossa voin ottaa jonkin komponentin irralleen ja käyttää sitä jossakin muussa sovelluksessa, tai liittää tähän sovellukseen jonkin ulkopuolella tehdyn komponentin. Sovelluksen skaalautuvuus mahdollisti sen, että jatkossa toimeksiantajan liiketoiminnan kasvaessa tätä sovellusta voitaisiin kasvattaa, eikä se törmäisi kovin helposti mihinkään pullonkaulaan kehityksessä.

2.4 Käytetyt sovelluskehityksen menetelmät

Pääasiassa projektien toteuttamiseen on karkeasti kahden tyyppistä lähestymistapaa, vesiputousmalli eli niin sanottu perinteinen projektimalli tai ketterät lähestymistavat. Vesiputousmalli soveltuu hyvin tarkasti ennustettaviin vaihe vaiheelta eteneviin projekteihin. Tässä mallissa seuraavaan vaiheeseen siirrytään vasta edellisen vaiheen toteutuessa. Tämä malli toimii hyvin esimerkiksi projekteihin, missä tiedetään jo alusta lähtien mitä halutaan. Ketterässä kehitysmenetelmässä projektia tekevä taho koostuu tiimistä, joka tekee iteratiivisesti ohjelmistoprojektia. Projektia tehdään 2-4 viikon sykleissä, joiden välissä tiimi esittelee tuotoksen ja käy neuvottelut halutuista seuraavista askeleista ja muutoksista. Ketterä menetelmä soveltuu hyvin esimerkiksi silloin, kun projektin halutusta lopputuloksesta ei ole selkeää kuvaa tai projektin aikana voi tulla muutoksia. (Thinking Porfolio 2016.)



Kuva 1. Menetelmämallit (mukaillen Thinking Portfolio 2016)

Minun oli helpompi valita tämän toimeksiannon malliksi vesiputousmenetelmä, sillä toimeksiannolla oli selkeä tavoite ja selkeä kuva sovelluksesta, joka sillä toteutettiin. Kuten kuvasta 1 voi havaita, sovelluskehitys jaettiin eri osiin, joka päättyy sovelluksen julkaisun jälkeiseen ylläpitoon. Projektin alussa kävimme asiakasyrityksen kanssa läpi tavoitteita ja malleja sovelluksen ulkoasuun. Tämän jälkeen suunnittelin toimeksiannon etenemistä, jonka ansiosta pystyin havainnollistamaan toimeksiantajalle projektin etenemisen vaihe vaiheelta. Mallin valitseminen ja suunnitelmassa pysyminen helpottivat työn toteuttamista, koska toteutin koko projektin itse. Pystyin keskittymään aina tiettyyn vaiheeseen ja pyrin välttämään ylimääräisten asioiden tekemistä, vaikka niiden teko olisi ollut juuri sillä hetkellä helppoa.

3 Sovelluksen yleiskuvaus, rakenne ja vaiheet

Aloitin työn suunnittelun pohjustuksen siihen tietoon, mikä oli minulle entuudestaan tuttua. Sovelluksen rakentamiseen käytettiin sovellusarkkitehtuurin periaatteita, jotka ovat suosittuja Angular-ohjelmistokehyksillä sovellusta tehdessä. Sovelluksen yleisrakenne noudatti usein verkkosivuilla käytettyjä verkkosovelluksia, jolla sen käyttöä pyrittiin selkeyttämään.

3.1 Sovelluksen rakenne ja arkkitehtuuri

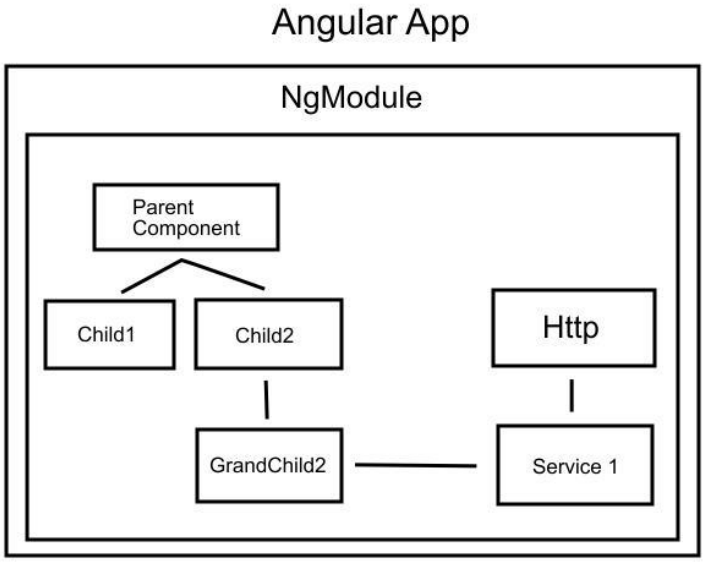
Ohjelmistosysteemin arkkitehtuuri on sen rakentajien päättämä muoto. Muoto on jaettu komponentteihin, näiden kyseisten komponenttien järjestykseen ja miten kyseiset komponentit kommunikoivat keskenään. Arkkitehtuurilla on hyvin vähän tekemistä sovelluksen toimivuuden kanssa. Markkinoilla on monia sovelluksia, joiden arkkitehtuuri on heikko, mutta sovellus toimii aivan hyvin. Arkkitehtuurin päätehtävä on kannatella sovelluksen elinkaarta. Hyvä arkkitehtuuri tekee ohjelmasta ymmärrettävän, helpommin kehitettävän ja ylläpidettävän. (Martin 2018, 136-137.)

Angularin arkkitehtuurin kehittämiseen ajateltu tapa on luoda moduuleita, joista tärkeimmät ovat ydinmoduuli, ominaisuusmoduulit ja jaetut moduulit. Ydinmoduulit ovat tarkoitettu käytettäväksi vain kerran sovelluksessa. Ominaisuusmoduulit sisältävät sovelluksen kunkin eri ominaisuuden komponentit. Jaetut moduulit ovat niitä ominaisuuksia, joita käytetään useammassa paikassa sovelluksessa tai jopa aivan muissa sovelluksissa. (Braga 2019.)

Arkkitehtuurisesti ominaisuuksien jaottelu selkeyttää sovelluksen tekoa ja skaalautuvuutta. Jos jatkossa tarvitsee muokata sovellusta tai tehdä jokin uusi sovellus, joka käyttää samanlaista ominaisuutta, voidaan tämä ominaisuus ottaa tästä sovelluksesta. Tämä on hyvä vaihtoehto perinteiselle tavalle luoda verkkosovellusta, jossa erityyppiset sovelluksen osat, kuten komponentit, tyylit tai mikä tahansa on omissa kansioissaan. Jokainen ominaisuus tämän jälkeen jaotellaan moduuleihin, jotka ladataan Lazy Loading -periaatteella eli vain, kun ohjelmisto tarvitsee sitä. (Braga 2019.)

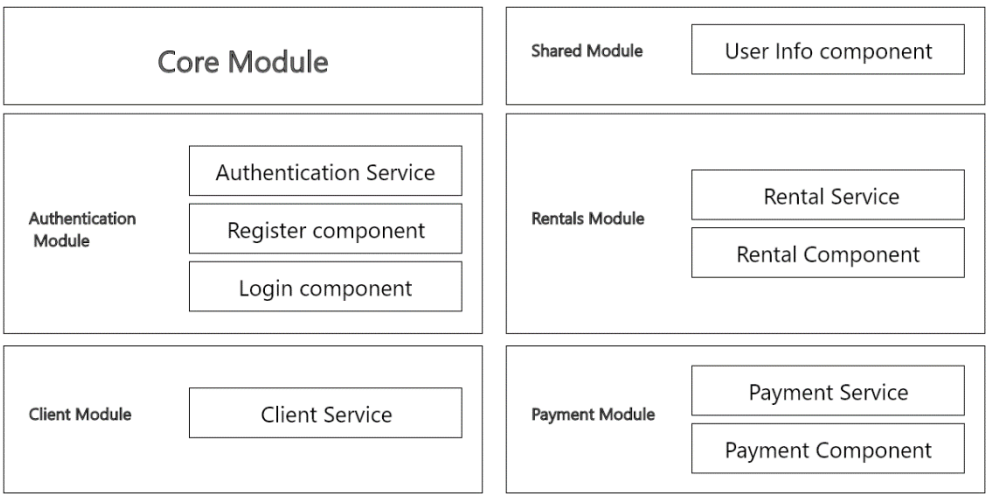
Sovellus voidaan jakaa useampaan moduuliin, joista jokainen moduuli sisältää tietyn funktionaalisuuden, jonka se paljastaa tai piilottaa. Moduulit voidaan ladata vasta siinä vaiheessa, kun niitä tarvitaan, jolloin käyttäjän ei tarvitse ladata kuin mahdollisimman vähän tarvittavaa koodia. Mitä vähemmän koodia sovelluksen tarvitsee ladata, sen nopeampi sitä on käyttää, kun turhaa ladattavaa ei ole. Tämä on erityisen tärkeää mobiilisovelluksissa, varsinkin jos niitä käytetään huonon kuuluvuuden alueilla. (Fain & Moiseev 2017, 95-97.)

Komponentit ovat sovelluksen pienimpiä osia, kuten yksittäisiä tiedostoja. Komponentteja voidaan yhdistellä yhdeksi ajettavaksi tiedostoksi tai niitä voidaan yhdistellä yhdeksi kokoelmaksi. Riippumatta siitä, miten niitä käytetään, hyvin suunnitellut komponentit ovat aina suunniteltu yksittäin kehitettäviksi, jolloin kehittäjän ei tarvitse keskittyä kuin yhteen komponenttiin kerrallaan. (Martin 2018, 96.)



Kuva 2. Sample architecture of an Angular app (mukaillen Fain & Moiseev 2017, 9)

Käytin sovelluksen arkkitehtuurin suunnittelussa niin sanotusti ominaisuus ensin ajattelutapaa. Pyrin noudattamaan moduuleiden suunnittelussa kuvan 2 mukaista rakennetta. Sovelluksessa oli ydinmoduulin lisäksi kahdenlaisia moduuleita: ominaisuusmoduuleja ja jaettuja moduuleja. Kuten kuvasta 3 voi huomata, moduuleiden sisältä löytyi komponentteja sekä Service-komponentti, jotka kommunikoivat keskenään. Jaetun komponentin User Info-komponenttia käytettiin sekä Client-moduulissa että Authentication-moduulissa.



Kuva 3. Sovelluksen arkkitehtuurinen rakennesuunnitelma

Sovellus muodostuu seuraavista ominaisuusmoduuleista:

- Valtuuttaminen, joka sisältää kirjautumisen ja rekisteröinnin
- Asiakastiedot
- Kotikomponentti
- Vuokraukset

Sekä seuraavista jaetuista moduuleista:

- Käyttäjätiedot
- Navigointikomponentti

Sovelluksen eri ominaisuuksien jaottelu moduuleihin helpotti sovelluksen tekoa. Oli huomattavasti helpompaa työstää pelkästään yhtä moduulia kerrallaan ja siihen luotuja komponentteja, kun ei tarvinnut huolehtia vielä niiden toimivuudesta toisten moduuleiden kanssa. Lisäksi tein jokaisen moduulin sisälle oman reititin moduulin, jolla hallinnoitiin moduulin sisällä tapahtuvaa liikennettä eri html -osoitteisiin. Kun olin saanut rakennettua perustoiminnallisuuden moduulin sisällä oleviin komponentteihin, siirryin luontevasti seuraavaan moduuliin ja ominaisuuteen, joka saattoi tarvita aikaisempaa moduulia. Tämän seikan vuoksi minun olikin paras aloittaa käyttäjän luomisesta ja tunnistautumisesta, joten ensimmäisenä työstin rekisteröinnin ja kirjautumisen. Näiden jälkeen saatoin käyttää rajapinnasta tuotuja testikäyttäjätietoja sovelluksen seuraavien moduulien kehittämiseen.

Luodessani tarvittavia komponentteja, niin minulla oli käytössäni jo niissä esitettävät tiedot. Esimerkiksi käyttäjätietokomponentissa minulla oli käytössäni jo nimi, sähköposti ja osoitetiedot, jotka minun piti vain asetella näkyviin sivulle. Halusin tehdä käyttäjätietoja käsitteleville sivuille mahdollisuuden helppoon muuttamiseen, joten tämän vuoksi erittelin käyttäjätiedot erilliseen jaettuun komponenttiin, jota pystyin käyttämään sellaisenaan haluamillani sivuilla. Tätä komponenttia ei kuitenkaan käytetä kuin parissa kohtaa sovelluksessa, niin voisi väittää tämän olevan vähän turhan paljon koodia niinkin pieneen asiaan, mutta halusin olla toistamatta samaa koodia yhä uudestaan.

3.2 Sovelluksen ulkoasun suunnittelu ja luonnostelu

Käyttöliittymät ovat käyttäjän pääsy sovellukseen. Käyttöliittymän suunnittelu on taito, joka on tärkeä osa käyttökokemuksen suunnittelussa. Käyttäjät ovat nopeita tekemään päätöksiä ulkoasun mukaan. Suunnittelijat keskittyvät luomaan käyttöliittymiä, joita käyttäjät kokevat selkeinä ja tehokkaina. Sen täytyy luoda illuusio, että käyttäjät eivät käytä laitetta vaan he yrittävät saavuttaa jotain suoraan ja niin vaivattomasti kuin mahdollista. (Interaction Design Foundation s.a.)

Käyttöliittymää ei kannata sekoittaa käyttökokemukseen. Käyttöliittymä pyrkii olemaan suunnittelun pinta ja tunne. Graafisista käyttöliittymistä pyritään luomaan esteettisesti miellyttäviä ja luonnollisesti toimivia. Elementit kuten napit toimivat kuten niiden voisi arvata toimivan ja kosketustoiminnot toimivat kuten ne toimivat kaikkialla. Ikonien täytyy olla selkeitä ja käyttöliittymien puhtaita. Suunnittelijan täytyy huomioida käyttäjän silmän kulkemista sovelluksen hierarkian läpi. Asettelun täytyy ohjata käyttäjän silmää ja kiinnittää huomio haluttuihin kohtiin. (Interaction Design Foundation s.a.)

Tein sovelluksen ulkonäöstä ensin karkean käyttöliittymäluonnostelun, joka antoi toimeksiantajalle ajatuksen siitä, minkälainen sovellus tuli olemaan. Kuvasta 4 voi havaita ensimmäiset versiot sovelluksen käyttöliittymän ensimmäisistä elementeistä ja yleisestä tyylistä.

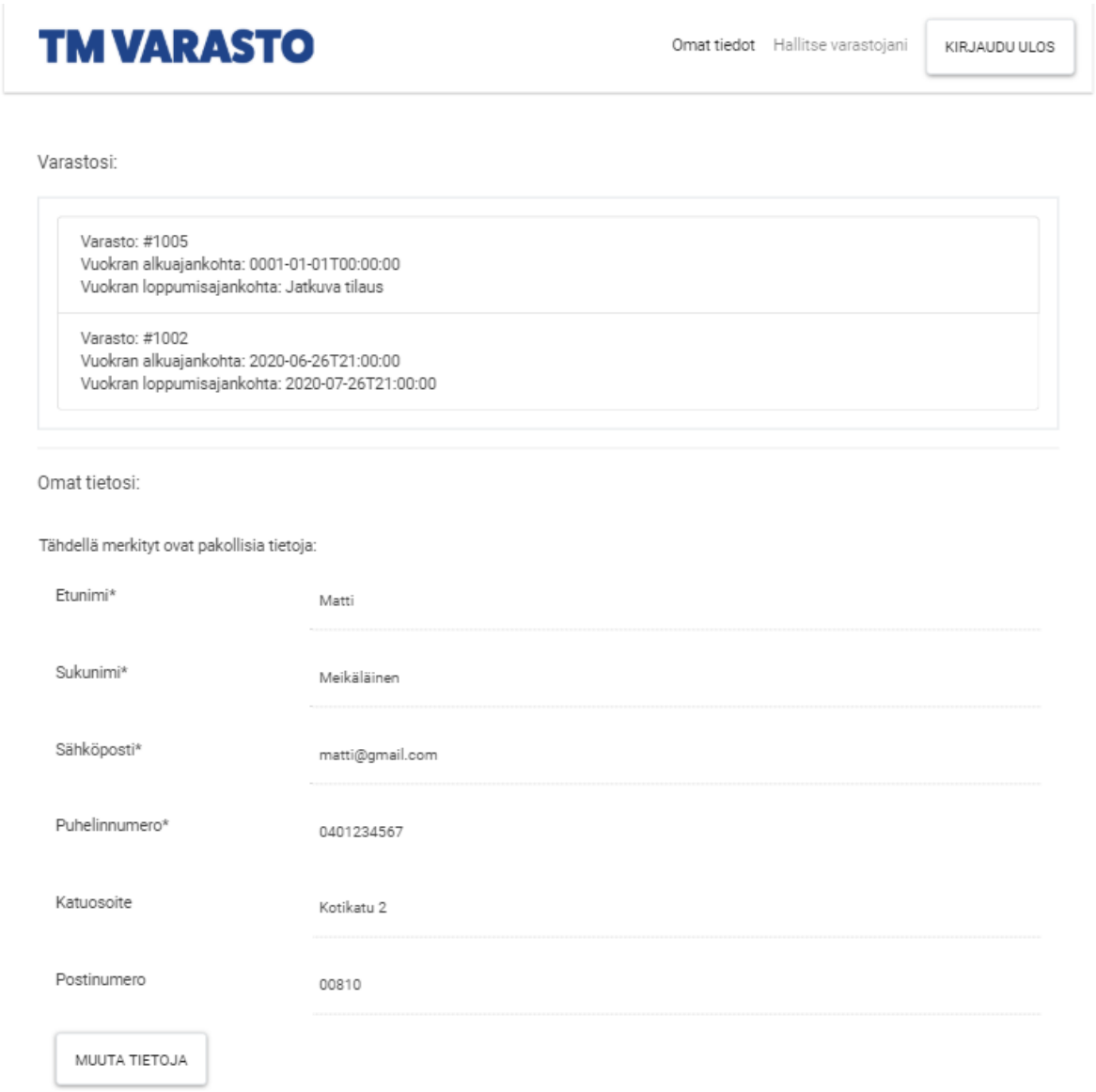
Kuva 4. Sovelluksen ensimmäinen käyttöliittymäsuunnitelma

Toimeksiantaja pääsi näin ollen mukaan suunnitteluun ja antamaan oman palautteensa sovelluksen ulkonäöstä. Tällä tavoin pystyin esittelemään omia ajatuksiani sovelluksen käytöstä ja toimeksiantajani pystyi antamaan kommentteja esimerkiksi käytetyistä väreistä ja tyyleistä. Suunnittelu ei ollut toimeksiantajani vahvoja puolia, joten he antoivat minulle vapaammat kädet tehdä sovelluksen suunnittelua. Tämä sopi minulle hyvin, mutta halusin pitää läheisen kontaktin suunnittelussa toimeksiantajan kanssa. Tällä tavalla he tietäisivät kaikissa vaiheissa, minkälainen heidän tilaamastaan sovelluksesta tulisi.

Luonnostelussa käytin Adobe XD käyttöliittymäsuunnittelu sovellusta, jolla pystyin helpommin asettelemaan luonnostelemani osaset paikoilleen ja tuomaan kuvat edelleen lähetettäväksi. Sovelluksen käyttöliittymän hahmotteleminen ja suunnitteleminen oli tärkeitä

vaiheita sovelluksen teossa, sillä niiden avulla pystyin havainnollistamaan toimeksiantajalle, kuinka sovellus tulee. Asettelin luonnostelmaan karkean toiminnallisuuden, esimerkiksi painikkeista pystyi liikkumaan seuraavaan diaan, joka näytti mikä osio sovelluksesta tulisi esiin käyttäjän painaessa kyseisestä painikkeesta.

Ensimmäisten palautteiden ja korjausten jälkeen aloitin rakentamaan sovelluksen komponentteja. Sovelluksen lopullinen ulkoasu muodostui vasta loppuvaiheilla, koska toiminnallisuuden toimintaan saaminen on tärkeämpää kuin ulkoasun viimeistely, joka voi omalta osaltaan viedä hyvinkin paljon aikaa. Vielä ennen käyttöönottoa piti löytää pienemmätkin ongelmakohdat, kuten kuvasta 5 voi huomata esimerkiksi, että vuokrausten ajat eivät ole kovin selkeitä käyttäjälle.



Kuva 5. Sovelluksen ensimmäisen käyttöönotetun version rekisteröintilomake

Sovelluksen ensimmäiset versiot, jotka otettiin käyttöön, olivat jo kehittyneet huomattavasti suunnitteluversioista, kuten liitteenä olevasta kirjautumis- ja rekisteröitymislomakkeesta voi huomata (liite 1). Eniten työtä vaati liitteenä 2 oleva palvelun vuokrausosio sovelluksesta. Tähän osioon liitettiin maksupalvelut ja selkeämpi varastojen varauslistaus.

3.3 Sovelluksen osien luonti

Ohjelmointirajapinta eli rajapinta mahdollistaa tiedon siirtämisen kahden eri ohjelman tai sovelluksen välillä. Rajapinnoilla voidaan hakea tietoa tai siirtää sensitiivistä informaatiota vaikkapa pankkitilien välillä. Rajapintojen tarve on lisääntynyt sitä mukaa, miten paljon palveluita on kehitetty. Ne tekevät sovellusten käyttämisestä miellyttävää ja lisäävät siten käyttökokemusta. (Haglund 2018.)

CLI on lyhennelmä sanoista Command Line Interface. Se on tekstipohjainen käyttöliittymä tiedostojen näyttämiseen ja hallinnointiin. Komentoriviä on käytetty tietokoneen käyttämiseen jo ennen hiiren ja graafisen käyttöliittymän lisäämistä tietokoneisiin. Komentorivillä voidaan ajaa ohjelmia ja erilaisia komentoja ohjelmien sisällä. CLI voi olla kuitenkin vaikea hahmottaa ja käyttää, sillä se ei opasta käyttäjäänsä mitenkään, mutta toisaalta on nopea ja tehokas tapa joidenkin prosessien ajamiseen. (Rouse 2018.)

Tunniste eli englanninkielinen token, on pieni tekstipohjainen tiedon pala, jolla itsenään ei ole tarkoitusta tai käyttöä, mutta yhdistettynä oikeanlaiseen tunnistekäytäntöön siitä tulee merkittävä sovelluksen turvallisuustekijä (Auth0 2020). Tunnisteen käyttö kuuluu omistajuuspohjaiseen kategoriaan, missä tunnistautuminen tehdään jollakin minkä omistat. Käyttäjän tarvitsee omistaa tarvittava tunniste, jotta hän saa pääsyn tiettyyn tietoon. Tämä voidaan yhdistää salasanapohjaiseen tunnistekäytäntöön paremman tietoturvan varmistamiseksi. (SolarWinds 2020.)

Aloitin sovelluksen käytännön toteutuksen ominaisuus kerrallaan. Yhden ominaisuuden valmistuessa ohjelmoinnin aikana, niin pystyin testaamaan sitä samaan aikaan rajapinnan kanssa. Testasin toiminnallisuutta pintapuolisesti aina, kun sovelluksen jokin osa otti yhteyttä rajapinnan avoimiin testauslähteisiin. Minulla oli valmis arkkitehtuurinen rakennelma, joten pystyin käyttämään Angular CLI komentoja ensin eri moduulien luomiseen ja sen jälkeen erillisten komponenttien luomiseen moduulien sisällä. Angular CLI sisältää useita työtä helpottavia komentoja ja komennon jälkeenkin se kysyy automaattisesti lisätietoa, jos sitä ei ole itse muistanut lisätä komentoon.

Ensimmäisten osien luomisen jälkeen ja yhteyden testaamisen jälkeen alkoi työ komponenttien yhdistelemisessä. Jokaisella komponentilla oli oma Service-komponentti, jonka

tehtävänä oli hoitaa informaation sisältäminen ja yhteydenpito rajapintaan. Kaikkein tärkeimmässä asemassa oleva valtuutuskomponentti oli mukana kaikissa muissa komponenteissa. Ohjelmiston tietoturvallisuuden takia ohjelman täytyi sisältää yksi komponentti, joka piti sisällään tunnistautumista varten luodun tunnisteen. Tunniste, joka on ohjelmisto-olio, pitää sisällään käyttäjän tunnistamiseen olevan informaation. Itse tunnistautuminen tapahtuu palvelimen rajapinnassa, joten sen muuttaminen manuaalisesti ei vaaranna tietoturvaa. Tämä tunniste tallennettiin käyttäjän selaimen työmuistiin, jossa se säilyi ennalta määritellyn ajan. Kun käyttäjä kirjautui sisään sovelluksen, niin hän sai tunnisteen, jonka avulla käyttäjän ei tarvinnut sen voimassaolon aikana kirjautua uudestaan, ellei hän kirjautunut tarkoituksella ulos.

Sovelluksen eri komponenttien latauksessa käytettiin laiskaa latausmuoto tekniikkaa, joka mahdollisti mahdollisimman pienen latausmäärän sovellusta käytettäessä. Jos asiakas tutustui vain rekisteröintiin, hänen selaimensa ei tällöin tarvinnut ladata koko sovelluksen koodia käyttöönsä vaan se latasi vain sen tarvitsemansa osion. Näin ollen latausmäärät pysyivät mahdollisimman pieninä ja käyttäjien ei tarvinnut odotella juuri ollenkaan.

3.4 Kolmannen osapuolen osat ohjelmassa

Ohjelmistokehittäjän pitäisi välttää käyttämästä aikaa jo ratkaistujen asioiden uudelleen ratkaisemiseen. Ohjelmistokehittäjänä tähän käytetään usein kolmannen osapuolen kehittämiä ohjelmistokirjastoja. Niiden käyttämisestä käydään jatkuvaa keskustelua kuitenkin, sillä ei ole epätavallista nähdä skeptisiä kokeneita kehittäjiä, jotka kieltäytyvät käyttämästä niitä. Kolmannen osapuolen ohjelmien käytöllä voidaan säästää aikaa ja hyödyntää jo ennalta testattua koodia. Ohjelmistokirjastoja tehdäänkin ratkaisemaan jo olemassa olevia ongelmia. Ne kannustavat kirjoittamaan modulaarista koodia, koska ohjelmiston eri osat ovat paloiteltu paremmin hyödynnettäviin osiin. (Papadopoulos 2018.)

Kolmannen osapuolen kirjastojen hyödyntämisen huonoihin puoliin kuuluu niiden riippuvuudet. Jos käytät jotakin kirjastoa, niin silloin oma koodisi on riippuvainen siitä kirjastosta. Tämän kirjaston päivittäminen on omalla vastuullasi, etkä voi olla koskaan täysin varma, toimiiko käyttämäsi koodi tulevaisuudessa. Ongelmaksi voi muodostua kirjaston vanhentunut koodi. Jos kyseistä koodia ei kukaan päivitä, voi se jossain vaiheessa aiheuttaa ongelmia tekniikoiden kehittyessä. Jotkut kirjastot aiheuttavat keskenään ongelmia, jolloin ongelman löytäminen voi olla hankalaa. Kolmannen osapuolen koodin käyttäminen voi altistaa oman ohjelmasi haavoittuvalle koodille. (Papadopoulos 2018.)

Jos on kiinnostunut web-ohjelmiston kehittämisestä, on hyvin todennäköisesti kuullut Bootstrapista. Bootstrap on kaikkein suosituin ohjelmistokehys responsiivisten käyttöliittymien kehittämiseen. Jos on kuitenkin vain tehnyt verkkosivuja pelkästään perustekniikoilla, voi ohjelmistokehyksen käyttäminen tuntua oudolta. Sen käyttäminen vähentää huomattavasti toistuvan koodin tekemistä, nopeuttaa prototyyppien testausta ja sisältää jo testatut ominaisuudet selainten väliseen kehittämiseen. (Rascia 10.11.2015.) Bootswatch on kokoelma erilaisia teemapaletteja, joita voi ladata ilmaiseksi ja liittää omaan Bootstrap-projektiin. Bootstrap on Twitterin kehittäjien tekemä ohjelmistokehys, mutta sillä tehtyjen sivujen ongelmaksi on muodostunut yhtenäisyys. Bootswatchin avulla voi vaihtaa teemaa ilman, että tyylitiedostoihin tai koodiin tarvitsee tehdä yhtään muutosta. (Park 2012.)

JSON Web Token ei ole pintapuolisesti mitään muuta kuin pitkä lista satunnaisia kirjaimia ja numeroita. JWT on kuitenkin paljon enemmän kuin se. Se on turvallisen tiedon lähetyksen määrittely. JWT:n satunnaisen näköisen rivin muodostaa kolme osaa, joita erottaa piste. Nämä kolme osaa muodostavat oman roolinsa, tehden siitä erittäin vaikean, ellei jopa mahdottoman, kenenkään ulkopuolisen käsitellä. Ensimmäinen osa on otsikko, joka sisältää tunnisteen allekirjoituksessa käytetyn algoritmin. Toisessa osassa on informaatio, joka halutaan sisällyttää tunnisteeseen. Tunnisteen informaatiot muodostavat väitteitä, joilla väitetään jonkin olevan totta. Sen jälkeen käytetään allekirjoitettua ja suljettua tunnistetta todentamaan nuo väitteet todeksi. Kuka tahansa, jolla on käytössään tunniste, voi lukea sen sisällön koska tahansa käyttämällä mitä tahansa ohjelmaa, joka voi purkaa koodauksen. Se, mikä tekee tunnisteesta turvallisen, on allekirjoitus. Allekirjoitus on salattu siten, ettei sitä voi purkaa. Rajapinta voi tunnistaa salauksen ja tunnistaa sen avulla käyttäjän, kun sovellus on yhteydessä rajapintaan. (Chenkie 2019, 118-121.)

Kolmannen osapuolen kirjastojen käyttäminen helpottaa eri kehitystehtävissä, niinpä päädyin hyödyntämään joitakin suosittuja ja turvallisiksi luokiteltuja lisäosia. Jotta ohjelmiston kehittämisessä ei menisi valtavan kauan turhaa aikaa, eikä minun tarvinnut luoda mitään täysin tyhjästä, käytin tyylittelyyn jo Bootstrap-liitännäistä. Tämän liitännäisen avulla pystyin käyttämään jo valmiita käyttöliittymän komponentteja tehdessäni testattavaa versiota ohjelmistosta. Bootstrapin lisäksi käytin samaan sarjaan kuuluvaa Bootswatch-teemanhallintakirjastoa. Toimeksiantajan määrittelemä teema oli hyvin pitkälti tietyn väristä, ja Bootswatch-kirjastolla pystyin suoraan valitsemaan hyvin samankaltaista väriteemaa käyttävän teeman ohjelmalle.

Käytin sovelluksessa käyttäjän tunnistamiseen JSON Web Token -ohjelmistotekniikkaa, joten hyödynsin JWT Helper -ohjelmistokirjastoa helpottamaan tunnisteen käsittelyä. JWT Helperin avulla pystyin käsittelemään sovelluksen sisällä kyseistä tunnistetta, mutta se ei

kuitenkaan pystyisi murtamaan käyttäjän saamaa tunnistetta, sillä itse tunnistautuminen tapahtui palvelimen päässä. JWT-tunnisteeseen tuotiin erilaista tietoa, kuten käyttäjän tunnistautumisessa ja erittelyssä tarvittavan ID-tieto. Näiden kahden tärkeimmän lisäksi ohjelmassa käytettiin Font Awesome -kirjastoa, joka toi sovelluksessa käytettävät ikonit. Font Awesome -kirjasto oli monipuolinen ja sisälsi kaikki tarvitsemani kuvakkeet sovellukseen. Lisäksi tämä ikonikirjasto asentuu Bootstrap-kirjaston kanssa, niin minun ei tarvinnut sitä erikseen asentaa sovellukseen.

3.5 Sovelluksen käyttöönotto

Perinteisesti yritykset ja muut organisaatiot ylläpitävät omaa infrastruktuuriaan. Yrityksellä olisi oma verkkopalvelin sen omilla koneilla. Jos lisää tehoa tarvittaisiin, yrityksen täytyisi ostaa lisää koneita tai tehokkaampia osia. Yritys joutuisi maksamaan jollekulle koneiston hallinnoinnista ja tehokkaasta internet yhteydestä, jotta se voisi palvella asiakkaitaan. Vaihtoehtoisesti yritys voisi vuokrata tämän palvelun joltakulta toiselta yritykseltä. Pilvipalvelu toimii eri tavalla. Sen sijaan, että yritys investoi kalliisiin koneisiin tai vuokraa jonkun toisen datakeskusta, se voi maksaa pääsystä Microsoft Azureen tai muiden tarjoajien massiivisiin koneresursseihin. Näiden kautta se voi ylläpitää verkkopalvelimia, sähköposteja, tietokantoja, virtuaalisia koneita tai mitä tahansa vain yritys haluaakaan. Jos yritys tarvitsee lisää tehoja, sen ei tarvitse ostaa uusia laitteita. Pilvipalvelu jakaa työkuormaa ja tarvittaessa varaa lisää tehoja ottamalla useamman palvelimen käyttöön automaattisesti. Yritys maksaa vain käyttämästään tarvitun työn määrästä. (Hoffman 2018.)

Domain nimi on verkkosivun osoite, jonka ihminen kirjoittaa verkkoselaimensa osoitepalkkiin. Yksinkertaistettuna, jos verkkosivu on talo, niin domain on sen osoite. Koneiden osoitteet ovat normaalisti numeromuodossa, joka ei ole kovin ihmisluettava. Niinpä domain nimet on tehty ihmisten muistettavaksi ja käytettäväksi. Alidomain on taas päädomain nimen alainen, esimerkiksi videos.wpbeginner.com on alidomain wpbeginner.com domain nimestä. (Wpbeginner 2019.)

Sovelluksen käyttöversion valmistuessa olimme toimeksiantajan kanssa sopineet ensimmäisen toimivan version käyttöönotosta. Sovelluksessa ei ollut varsinaisesti mitään suuria kuvia tai elementtejä, niin sovelluksen koko pysyi erittäin pienenä. Sovellus siirrettiin käyttöönottoon Azuren siirtotyökaluilla palvelimelle, jossa siirron jälkeen sitä pystyi tarkastelemaan vielä ennen lopulliseen julkiseen esittelyyn ottamista.

Sovelluksen ensimmäinen versio sovittiin käyttöönotettavaksi ensimmäisessä muodossaan kesäkuussa 2020, eli heti kun siitä oli toimiva versio valmis. Sovellus siirrettiin Microsoft Azuren pilvipalvelussa toimivan tmvarasto.fi domainin alle. Sovellukselle oli tehty jo

alidomain, jolloin toimeksiantaja voi käyttää samaa SSL-sertifikaattia, joka oli heidän palvelimellaan alidomainissa toimivalla sovelluksella. Pilvipalvelussa toimivat rajapinta sekä tietokanta yhdistivät sovelluksen yhdeksi toimivaksi kokonaisuudeksi. Käyttäjät pystyivät käyttämään sovellusta menemällä toimeksiantajan palvelun verkkosivulle ja menemällä sieltä varaston vuokrauspalveluun joko linkkinappia painamalla tai kirjautumalla sivun navigointilinkistä.

4 Sovelluksen tietoturvallisuus

Sovellusten tietoturva on prosessi, missä tehdään sovelluksesta turvallisempaa etsimällä, korjaamalla ja vahvistamalla tietoturvaa. Suurin osa tästä tapahtuu sovelluksen kehitysvaiheessa, mutta sitä pitää tapahtua käyttöönoton jälkeen. Mitä aikaisemmin tietoturvaongelma löydetään ja korjataan, sitä parempi. Ohjelmistokehityksessä tapahtuu virheitä, sillä ohjelmoijatkin ovat vain ihmisiä. Aivan tavallinen tekstinsyöttö, jota ei tarkasteta voi olla hakkerille pääsy tietokannan tietoihin. (Strom, 2019.)

Sovelluskehityksessä tietoturvan huomiointi pitää ottaa jokaiseen vaiheeseen, joka usein koetaankin hidastavana tekijänä. Jälkeenpäin tietoturvan huomioiminen on kuitenkin usein kalliimpaa ja vaikeampaa toteuttaa. Sovellusten tietoturvallisuutta ei oppilaitoksissa tarpeeksi opeteta, minkä vuoksi uusilla ohjelmoijilla voi olla puutteelliset tiedot tietoturvallisen ohjelmoinnin osaamisessa. Varsinkin, kun sovellus koostuu useasta eri komponentista, joista vain osa on organisaation itsensä tekemiä. Muiden komponenttien tietoturvallisuuden taso kuitenkin vaikuttaa kokonaisuuden tietoturvallisuuteen. (Pietikäinen, 2013.)

4.1 Käyttäjätiedot ja niiden hallinta sovelluksessa

Tietoaineita käsitellään sovellusten kautta ja niiden luottamuksellisuuden säilyminen riippuu sovelluksen luotettavuudesta ja tietoturvallisuudesta. Sovellus pystytään luokittelemaan ja rakentamaan siten, että se sopii aiottuun tarkoitukseen käytettäväksi. Tietoturvavaatimuksia laadittaessa pitää ottaa huomioon luottamuksellisuus, eheys, saatavuus ja jäljitettävyys. Kehitettävästä sovelluksesta ja sen käyttötarkoituksesta riippuu, mitä näistä painotetaan. Jos sovellusta käytetään vain julkisen tiedon katseluun, ei käyttäjiä tarvitse tunnistaa. (Pietikäinen, 2013.)

Toimeksiantajani projektin alkaessa halusi, että sovelluksen kautta käsiteltiin joitakin ihmisten henkilökohtaisia tietoja sekä maksutapoja, jonka vuoksi kaikkein tärkein ominaisuus oli tietoturvallisuus ja käyttäjien tietojen hallinnointi. Tämän vuoksi sovelluksen tietoturvassa käytettiin huomattavasti aikaa ja sitä kehitettiin paremmaksi tämän toimeksiannon jälkeenkin. Sovelluksessa piti huomioida erityisesti Euroopan Unionin asettaman yleisen tietosuoja-asetuksen määräämät toimenpiteet henkilötietojen käsittelyyn.

Chenkien (2019, 20) mukaan ennen kuin käyttäjät voivat kirjautua sisään sovellukseen, täytyy vähintään seuraavat tietoturvallisuusasiat kunnossa:

- Rajapinta, jossa tehdään sisäänkirjautumiseen liittyvät tunnistautumiset ja päätökset
- Tietokanta, jossa ylläpidetään käyttäjien tietoja
- Tapa, jolla säilytetään käyttäjien salasanoja oikeaoppisesti

- Paikka, jossa käyttäjät voivat kirjautua sisään

Jos halutaan enemmän kuin vähimmäisvaatimukset kirjautumiseen, pitää lisätä:

- Salasanakäytäntö
- Salasanan vahvistus sähköposti
- Suojaus boteilta
- Mahdollisuus käyttäjän pyytää salasanan uusimista

Tässä sovelluksessa oli toteutettu Chenkien (2019, 20) mainitsemista kohdista kaikki vähimmäisvaatimukset, salasanakäytäntö ja mahdollisuus käyttäjän pyytää salasanan vaihtamista. Käyttäjillä oli sovellusta käyttäessään aina mahdollisuus poistaa omat tiedot tai omatoimisesti vaihtaa kaikki haluamansa tiedot.

Toimeksiantajan kanssa sovimme, että suojaus boteilta tultiin tekemään toimeksiannon jälkeen erillään, koska siihen vaikutti pilvipalvelimen asetukset, jossa sovellusta ylläpidettiin. Toimeksiantajan rajapinta oli jatkuvan kehityksen alla, jolloin tietoturvan täytyi olla yksprioriteetti. Käyttäjien luovuttamia tietoja säilytettiin rajapintaan yhteydessä olevassa tietokannassa eikä sovellus ollut missään vaiheessa suorassa yhteydessä tietokantaan.

Suunnitelmissa oli jatkossa asentaa asiakkaille mahdollisuus maksaa jatkuva tilaus pankki- ja luottokorteilla, jolloin tietoturvan täytyi olla erityisen hyvä sovelluksessa, etteivät tiedot valuneet ulkopuolisille tilausta lähetettäessä. Sovellus kuitenkin maksuvaiheessa yhdistettiin pankkien maksupalveluun, joissa pankkien tietoturva huolehti itse maksuliikenteen turvallisuudesta. Kaikissa tapauksissa, joissa asiakas saattoi paljastaa oman maksukykynsä tai salaisia numeroitaan, täytyi tietojen olla salattuja.

Pankkien tietoturvallisuuskaan ei ole välttämättä täysin kunnossa. Jopa 97 prosenttia maailman suurimpien pankkien verkkosovellukset ovat jollain tapaa haavoittuvaisia verkkohyökkäyksille. Vanhin paljastunut tietoturva-aukko on ollut tiedossa jo vuodelta 2011. Vain kolmen pankin pääverkkosivua sai kaikkein korkeimman A+ luokituksen sekä SSL salauksestaan, että verkkosivun turvallisuudesta. Nämä kolme pankkia ovat sveitsiläinen Credit Suisse, tanskalainen Danske Bank ja ruotsalainen Handelsbanken. (Immuniweb, 2019.)

4.2 Selainpuolen tietoturvallisuus

Kun rakennamme Angular-sovelluksia, kehittäjien pitää pystyä elämään sen tiedon kanssa, että julkisena sovelluksena kuka tahansa pystyy näkemään kaiken koodin mitä se

sisältää. Käyttäjien tarvitsee vain avata selaimensa kehittäjätyökalut nähdäkseen lähdekoodin. Tämän vuoksi mitään salaisuuksia ei voi olla selainpuolen sovelluksissa. (Chenkie 2019, 17-18.)

Käyttäjän kirjautuessa sovellukseen ja sovelluksen saadessaan tietoja palvelimelta käyttäjästään, on parempi, ettei niitä tallenneta missään muodossa sovellukseen eikä selaimeen. Ainoa, mikä kannattaa tallentaa sovellusta käytettäessä on JWT-valtuutus, jolla palvelin tunnistaa käyttäjän. Angular-sovelluksen yksi hyvistä puolista verrattuna tavalliseen verkkosivuun on sen kyky ladata tiedot kerran ja ylläpitää niitä käyttäjän käyttäessä sovellusta. Tämä tarkoittaa SPA sovelluksessa, että vaikka sovelluksessa on useampi kohta mitä käyttäjä voi tarkastella, palatessaan vaikkapa käyttäjän tiedot kohtaan tiedot ovat yhä paikallaan, eikä turhaa verkkoliikennettä tarvitse tapahtua.

4.3 Sovelluksen ja palvelimen välinen tietoturva

SSL-sertifikaatti on kirjainyhdistelmä sanoista Secure Sockets Layer. Sen avulla salataan yhteys nettisivun ja palvelimen välillä. Kun yhteys on salattu, ulkopuoliset eivät voi seurata tietojen vaihtoa koneiden välillä. HTTPS tarkoittaa taas lyhennettä Hypertext Transfer Protocol Secure, joka tarkoittaa salattua yhteyttä. Sivustolle asennettu SSL-sertifikaatti auttaa käyttäjän selainta tunnistamaan sivun, eikä se estä käyttäjän siirtymistä sivulle, ja käyttäjä näkee osoitepalkista sivun olevan turvallinen ja luotettava. (Hakukonemestarit, 2020.)

Erilaiset injektiohyökkäykset voivat tapahtua silloin kun joku onnistuu suorittamaan vihamielisen JavaScript-koodin sovelluksessamme. Nämä skriptit ovat yleensä suunniteltu etsimään ja varastamaan käyttäjän tunnuksia tai muuta sensitiivistä informaatiota. On monia tapoja, kuinka sellainen skripti voi päätyä sovellukseen. Siitä huolimatta miten hyökkäys tapahtuu, meidän täytyy suojata valtuutustamme. (Chenkie 2019, 137.)

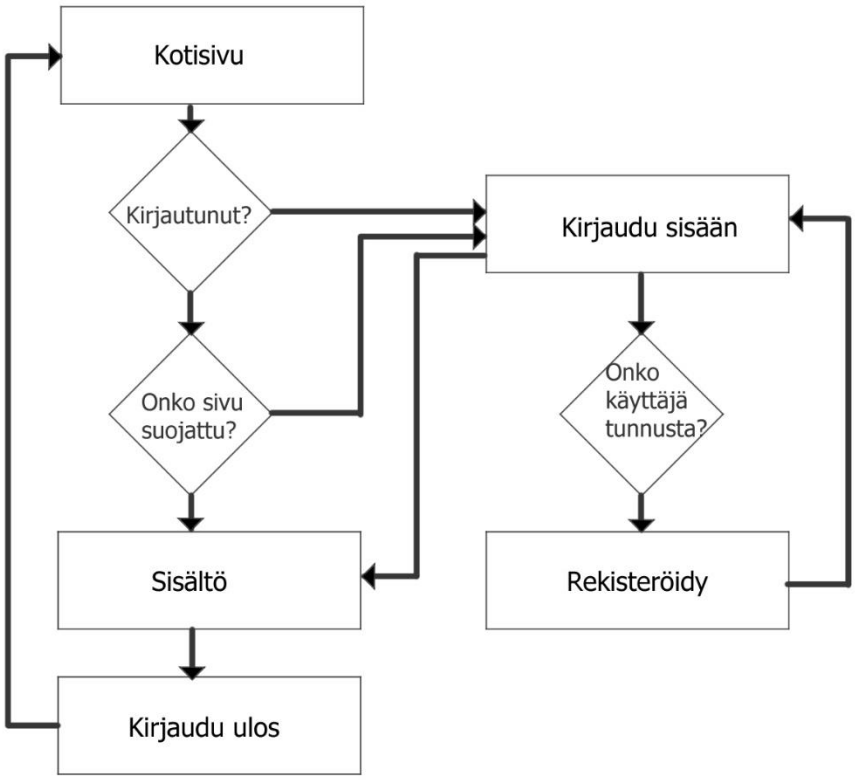
Toimeksiantajan omalla ohjelmistorajapinnan palvelimella käytettiin SSL-salausta tiedon liikkumiseen sekä sovelluksen sisällä käytettiin Angularin tietoturvallisia tapoja ottaa salattu https-yhteys yrityksen rajapintaan. Kaikki asiakastietojen ylläpitämiseen ja muuttamiseen käytettävät lomakkeet sisälsivät tarkistuksen, jotta lomakkeen eri kohtiin käyttäjä piti syöttää tietoa vain oikeanlaisessa muodossa. Lisäksi rajapinta tarkisti sinne syötetyt tiedot. Siinä tapauksessa, että joku ulkopuolinen jotenkin onnistuisi syöttämään käyttäjän lähettämään tietoon jotakin sinne kuulumatonta, palautti rajapinta virheilmoituksen ja esti käyttäjää suorittamasta vaarallista toimintoa.

Sovelluksen käyttäjän tunnistamiseen käytetty JWT-valtuutus oli suojattu salausavaimella, jonka vain palvelin tiesi ja joka oli kryptattu. Vaikka joku muuttaisikin sitä, tarvitsi heidän

silti tietää satunnaisgeneroitu salausavain sekä kryptausmenetelmä. Tämä valtuutus lähetettiin tietokutsun mukana tarkistusta varten. Palvelin tarkisti JWT-valtuutuksen sekä sen voimassaoloajan ennen kuin se antoi tietoja. Sovelluksessa käytettävät valtuutukset vanhenivat nopeasti. Sovellusta kehittäessä pyrimme hyödyntämään mahdollisimman useita lähteitä, jotta pystyisimme ennaltaehkäisemään tietoturvaongelmia.

4.4 Sovelluksen sisäiset käytetyt suojaus ja apumenetelmät

Angular sisältää pari omaa menetelmää käyttäjille sallittujen ja estettyjen reittien sallimiseen. Vaikka suurin osa käyttäjistä ei osaakaan nuuskia sovelluksen koodia, on erilaiset sovelluksen osat hyvä suojata. Angularilla on tähän tarkoitukseen suunnitellut reittisuojaimet, joilla voidaan estää käyttäjiä avaamasta sivuja, joihin vaaditaan tunnistautuminen. Selainpuolen sovelluksissa ei voi olla mitään salaista, koska käyttäjät voivat yhä katsoa koodista mitä kyseiset alueet sisältävät. Vaikka joku saisikin hakkeroitua itselleen pääsyn alueelle, johon heillä ei ole oikeuksia ilman valtuutuksia palvelinpuolelta, niin he näkevät vain rikkinäisen sivun. (Chenkie 2019, 140-141.)



Kuva 6. Workflow diagram of application (mukaillen Morioh s.a.)

Route guard on Angularissa käytetty komponentti, joka yhdistetään reittejä hallitsevaan reittimoduuliin. Route guardit palauttavat kyllä tai ei vastauksen ja ohjaa käyttäjän vastauksen mukaisesti oikealle sivulle tai estää kulun. Kun käyttäjä yrittää mennä kyseistä

reittiä pitkin, niin tämä suojakomponentti tarkistaa, onko käyttäjä oikeutettu käyttämään sitä sivua. (Chenkie 2019, 142.)

Vaikka selainpuolen sovelluksissa ei ole salaista tietoa, asetin ohjelman silti ohjaamaan käyttäjän kirjautumissivulle, jos käyttäjä ei ole kirjautunut tai kirjautuminen on vanhentunut. Käytin tähän Chenkien (2019) mainitsemaa Route Guard menetelmää. Sivun ollessa suojattuna kuvan 6 mukaisesti ja jos käyttäjä ei täytä ehtoja, käyttäjä palautetaan takaisin kirjautumisosioon. Tämän pystyi kokenut sovelluksen tutkija ohittamaan, mutta koska käyttäjien tiedot tuotiin vain ja ainoastaan käyttäjän ollessa kirjautunut, tämä lisäsi ainoastaan käyttäjäkokemusta. Käyttäjälle viestitettiin näin suoraan, että kirjautuminen vaaditaan kyseiseen toimenpiteeseen.

Verkko on pohjimmiltaan nopeaa edestakaista kutsujen ja vastausten tanssia. Yksisivuiset sovellukset tekevät kutsuja käyttäen XMLHttpRequestia. XMLHttpRequest on selaimiin sisäänrakennettu rajapinta, joka mahdollistaa http kutsu Javascript koodilla. Data, joka palautuu tällä kutsulla, tulee usein palvelinpuolen tietokannoista. XMLHttpRequestilla voi hakea tietoa monilla eri formaateilla. Tämä mekanismi mahdollistaa html-kutsuja sovelluksen sisällä ilman, että meidän tarvitsee tehdä koko sivun päivitystä ja on yksi tärkeimmistä asioista, mitkä tekevät yksisivuisista sovelluksista sen mitä ovat. Voimme tehdä muokkauksia http-kutsuun ja lisätä siihen parametrejä. (Chenkie 2019, 161; TutorialsPoint s.a.)

Http Interceptor tekee tämän automaattisesti jokaisella kutsulla. Kun kutsu lähetetään johonkin, se otetaan ensin vastaan http interceptorissa ja sen jälkeen käsitellään sen sisällä, ennen kuin se jatkaa joko sellaisenaan tai kloonattuna muutetulla sisällöllä. Http-kutsut ovat muuttumattomia tarkoituksella ja se on yksi tapa pitää sovelluslogiikka ennaltaarvattavana. Angular kiertää tämän tekemällä kloonin kutsusta, jota pystytään muuttamaan ja tämän jälkeen lähetetään klooni alkuperäisen sijaan. (Chenkie 2019, 161-164.)

Helpottaakseni hieman koodin kirjoitusta jokaisen menetelmän kohdalla, jotka käsittelivät tiedon kulkua, käytin http interceptoria lisäämään jokaiseen palvelimelle menevään kutsuun turvallisuusvaltuutuksen. Tämän ansiosta pystyin hieman vähentämään koodia kutsuja käsittelevissä metodeissa. Tein tarkistuksen sisäänpäin tulevissa vastauksissa tapauksissa, joissa käyttäjä ei ole tunnistautunut eli 401 virheilmoituksessa. Tällöin käyttäjä ohjataan automaattisesti kirjautumissivulle.

5 Pohdinta

Sovelluksen kehittäminen alusta loppuun oli erittäin avartava ja kiinnostava kokemus. Alkusuunnitelmien tekemisestä lopullisen työn valmistamiseen oli monipuolinen matka, vaikka sovelluksen kehitys jatkui tämän toimeksiannon jälkeen. Lopputuloksena syntynyt sovellus noudattaa niitä periaatteita, joista toimeksiantajan kanssa oli alussa sovittu. Sovelluksen ulkoasu muuttui työn edetessä, mutta ne olivat kuitenkin pääosin kosmeettisia muutoksia. Näin ollen projektissa käytetty vesiputousmenetelmä toimi suurin piirtein niin kuin olin sen suunnitellut. Syntyneessä tuotoksessa asiakas pystyi rekisteröitymään ja kirjautumaan sisään, varaamaan itselleen varaston sekä muokkaamaan omia tietojaan. Tästä rekisteröitymisestä ja varauksista lähti tieto toimeksiantajalle, joka sen jälkeen pystyi hoitamaan liiketoiminnalliset tehtävät uuden asiakkaan kohdalla.

Työskentelyni oli varsin omatoimista koko projektin ajan. Alkuvaiheen suunnittelussa olin yhteydessä toimeksiantajaani, mutta sain aika vapaat kädet sovelluksen toteuttamiseen. Minua tämä ei haitannut ja halusin silti pitää toimeksiantajan kartalla siitä missä mennään työn eri vaiheissa. Projektissa ei sinänsä ollut mitään varsinaisen monimutkaista, joka olisi vaatinut jatkuvaa neuvottelua toimeksiantajan kanssa. Sovellus sisälsi hyvin pitkälle samoja periaatteita kuin monissa muissakin verkkosovelluksissa, joita olin aikaisemmin tehnyt työssäni. Sovelluksen yhdistyessä rajapintaan, sen toiminnallisuus muistutti omalla tavallaan perinteisiä käyttäjäpohjaisia verkkosovelluksia. Tämä oli ihan hyvä asia, sillä ihmiset ovat tottuneet käyttämään tietyn tyyppisiä sovelluksia, jolloin tämän sovelluksen käyttö olisi tutumpaa loppukäyttäjälle.

Lähteissä pyrin esittelemään lukijalle teknisempien termien sisältöä. Sovellusta toteuttaessani pystyin tarkastelemaan tekemiäni valintoja lähteissä olevien artikkeleiden ja kirjojen kautta. Työn toteutuksen aikana varsinkin sovelluksen käyttämisessä sekä käyttäjän suojaamisessa turvauduin lähteissä esiteltyihin tapoihin, esimerkiksi Ryan Chenkien (2019) kirjassa käyttämät http interceptorit olivat minulle uusi tuttavuus ja hyvin mieluinen sellainen. Niiden käyttäminen vähensi jonkin verran koodin kirjoittamista. Myönnän tämän opinnäytetyön olevan tietotekniikkaan painottuvampi, eikä kaikki termit aivan täysin avaudu ICT-alan ulkopuoliselle lukijalle. Opinnäytetyöstä voisi kuitenkin saada näkemystä esimerkiksi tiimien vetäjälle tai esimiehille, miten verkkopohjainen sovellus syntyy.

Opin paljon teknisten valintojen tekemisestä ohjelmaa suunnitellessa. Ohjelmiston suunnittelu vaati huomattavasti paljon uuden opiskelua ja pidin itse koodin kirjoittamisesta. Se mikä minut yllätti erityisen paljon, oli tietoturvallisuuden huomioiminen sovellusta kehittäessä, vaikka tietoturvallisuutta ei varsinaisesti voi olla selainpuolen sovelluksissa. Näiden

asioiden huomioiminen vaati, että sovelluksessa ei saa paljastaa mitään palvelinpuolen salaisuuksista tai tietoturvaominaisuuksista, joka voisi aiheuttaa tietoturvaongelman.

Arkkitehtuurin pohdinta ja suunnittelu olivat minulle uutta tietoa ja tähän pyrin panostamaan enemmän. Työkokemuksestani oppien sanoa, että selkeä ohjelmistoarkkitehtuuri nopeuttaa ja selkeyttää tekijöiden ymmärrystä sovelluksesta ja varsinkin jos joku uusi tekijä liittyy tiimiin kesken projektin. Kokonaisvaltaisen arkkitehtuurin suunnittelemiseen tarvitsin kuitenkin lisää harjoitusta, jotta pystyisin suunnittelemaan sovellusarkkitehtuuria vielä tarkemmin. Tässä opinnäytetyössäni olisin voinut avata tätä puolta enemmän.

Angular ja siinä käytettävät tekniikat olivat minulle jo entuudestaan tuttuja työn puolesta, jonka vuoksi minun oli helppo ryhtyä tuottamaan sovellusta sen parissa. Olin tehnyt Angular-ohjelmistokehyksellä töitä jo muutaman vuoden. Tämä toimeksianto antoi minulle hyvät puitteet tehdä käyttöön tuleva sovellus täysin alusta alkaen ja opetella niitä puuttuvia taitoja, mitä sovelluksen eri osioiden tekemisessä tarvittiin. Työn kautta eri projekteissa tehdään eri asioita, mutta tässä toimeksiannossa sain opetella perinpohjaisesti kyseiseen ohjelmistokehykseen. Tämän ansiosta kokemus oli huomattavasti monipuolisempi kuin mitä normaalisti pääsen tekemään.

Tämän verkkosovelluksen kehitys oli aika perinteistä toimintaa ohjelmistokehityksen alalla, mutta minulle henkilökohtaisesti tämän työn tekeminen opetti hyvin paljon. Usein tämänkaltaisen sovelluksen tekemiseen olisi isoja tiimejä, joista eri henkilöt tekevät eri osa-alueita sekä rajapintaan että näiden itse sovellusten ohjelmointiin. Tällä kertaa vastasin itse sovelluksen eri osa-alueista, joten sain siitä hyviä näkemyksiä eri kehittämiskohteisiin tulevaisuudessa sekä miten tarkastelen työssäni vastaan tulevia projekteja.

Vastaavanlaisen työn tekeminen tulisi olemaan tulevaisuudessa helpompaa, mutta tämän sovelluksen tekeminen oli hyvin avartavaa. Sovelluksessa sekä sen ympärillä olevissa toiminnoissa oli toki vielä kehitettävää ja parannettavaa. Jatkossa sovellukseen lisättiin monipuolisemmat maksuominaisuudet ja eri paikkakunnan valinta. Olisin mielelläni kuitenkin mukana jatkossakin kehittämässä sovellusta ja osallistumassa siihen liitettyihin rajapinnan sekä pilvipalvelimen kehitykseen. Olin varsin tyytyväinen siihen, että pystyin tuottamaan tämän sovelluksen opinnäytetyönäni ja toivon, että voin jatkossakin toteuttaa vastaavanlaisia kokonaisuuksia. Sovelluksen jatkokehitys tulee kuitenkin vielä jatkumaan pitkään, jotta sitä voidaan kehittää ja optimoitua parempaan asiakaskokemukseen.

Lähteet

Auth0 2020. Token Based Authentication Made Easy. Luettavissa: https://auth0.com/learn/token-based-authentication-made-easy/. Luettu: 10.05.2020.

Backes M. 09.07.2019. HTML, CSS and Javascript Explained For Beginners. Luettavissa: https://www.codewall.co.uk/html-css-and-javascript-explained-for-beginners/. Luettu: 10.05.2020.

Bolton D. 2017. 5 Reasons to Use Visual Studio Code. Luettavissa: https://insights.dice.com/2017/05/26/5-reasons-use-visual-studio-code/. Luettu: 10.05.2020.

Braga A. 2019. Planning the Architecture of your Angular App. Luettavissa: https://itnext.io/planning-the-architecture-of-your-angular-app-a4840bfec13b. Luettu: 04.05.2020.

Chenkie R. 2019. Securing Angular Apps. Luettavissa: https://ryanchenkie.com/securing-angular-applications/. Luettu: 06.05.2020.

Digia Oy 2019. Mitä on digitaalinen liiketoiminta? Luettavissa: https://resources.digia.com/digitaalinen-liiketoiminta Luettu: 07.05.2020.

Fain, Y. & Moiseev A. 2017. Angular 2 Development with Typescript. Manning Publications Co. United States of America.

Fluin S. 25.12.2017. Why Developers and Companies Choose Angular. Luettavissa: https://medium.com/angular-japan-user-group/why-developers-and-companies-choose-angular-4c9ba6098e1c. Luettu: 14.05.2020.

Haglund J. 19.06.2018. API:t Käytännössä – Selkokielinen Katsaus. Luettavissa: https://www.alfame.com/blog/apit-kaytannossa-selkokielinen-katsaus. Luettu: 10.05.2020.

Hakukonemestarit.fi 2020. Mikä on SSL-sertifikaatti, ja miksi kotisivut tarvitsevat sellaisen? Luettavissa: https://www.hakukonemestarit.fi/blogi/mika-on-ssl-sertifikaatti-ja-miksi-kotisivut-tarvitsevat-sellaisen/. Luettu: 06.05.2020.

Halme A. 2018. Mikä on Single Page App ja mihin sitä käytetään? Luettavissa: https://city-devlabs.fi/single-page-app/. Luettu: 10.05.2020.

Hoffman C. 04.01.2018. What is Microsoft Azure, Anyway? Luettavissa: https://www.howtogeek.com/337961/what-is-microsoft-azure/. Luettu: 11.05.2020.

IBM 2020. What is software development? Luettavissa: https://www.ibm.com/topics/software-development. Luettu: 07.05.2020.

Immuniweb 2019. State of Application Security at S&P Global World's 100 Largest Banks. Luettavissa: https://www.immuniweb.com/blog/SP-100-banks-application-security.html. Luettu: 06.05.2020.

Interaction Design Foundation s.a. What is User Interface Design? Luettavissa: https://www.interaction-design.org/literature/topics/ui-design. Luettu: 28.05.2020.

Kempas K. 16.06.2017. Pienvarastojen kysyntä kasvaa, mutta moni yliarvioi tilantarpeensa ja päätyy maksamaan varastosta liikaa – Ihmiset eivät tahdo enää asua isossa asunnossa vain tavaroiden takia. Luettavissa: https://www.hs.fi/talous/art-2000005256395.html. Luettu: 12.02.2020.

Kyberturvallisuuskeskus 2019. Tietoturva. Luettavissa: https://www.kyberturvallisuuskeskus.fi/fi/toimintamme/saantely-ja-valvonta/tietoturva. Luettu: 07.05.2020.

Laine T. 23.09.2015. Mitä markkinoijan tulee ymmärtää web-ohjelmoinnista. Luettavissa: https://www.dagmar.fi/verkkopalvelukehitys/mita-markkinoijan-tulee-ymmartaa-web-ohjelmoinnista/. Luettu: 14.05.2020.

Martin R. C. 2018. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall. United States of America.

Morioh s.a. Ionic 4 and Angular 7 Tutorial: Securing Pages using Route Guard. Luettavissa: https://morioh.com/p/2192429f080c. Luettu: 20.05.2020.

Papadopoulos A. 2018. Should developers use thrid-party libraries? Luettavissa: https://www.scalablepath.com/blog/third-party-libraries/. Luettu: 10.05.2018.

Park T. 14.02.2012. Introducing Bootswatch. Luettavissa: https://thomaspark.co/2012/02/introducing-bootswatch/. Luettu: 10.05.2020.

Pietikäinen S. 2013. Sovelluskehityksen tietoturvallisuus ja hallinnointi. Luettavissa: https://www.vahtiohje.fi/web/guest/sovelluskehityksen-tietoturvallisuus-ja-hallinnointi. Luettu: 05.05.2020.

Progress Sitefinity s.a. Deploy projects to Azure App Services. Luettavissa: https://www.progress.com/documentation/sitefinity-cms/run-your-websites-on-azure-app-services. Luettu: 20.05.2020.
Rascia T. 10.11.2015. What is Bootstrap and How Do I Use It? Luettavissa: https://www.taniarascia.com/what-is-bootstrap-and-how-do-i-use-it/. Luettu: 10.05.2020.

Red Hat s.a. What is an IDE? Luettavissa: https://www.redhat.com/en/topics/middleware/what-is-ide. Luettu: 28.05.2020.

Rouse M. 2018. Command Line Interface (CLI). Luettavissa: https://searchwindowsserver.techtarget.com/definition/command-line-interface-CLI. Luettu: 10.05.2020.

Singh V. 2020. What is Frameworks? Luettavissa: https://hackr.io/blog/what-is-frameworks. Luettu: 10.05.2020.

SolarWinds MSP 06.02.2020. How Does Token-Based Authentication Work? Luettavissa: https://www.solarwindsmsp.com/blog/how-does-token-based-authentication-work. Luettu: 10.05.2020.

Strom D. 2019. What is application security? A process and tools for securing software. Luettavissa: https://www.csoonline.com/article/3315700/what-is-application-security-a-process-and-tools-for-securing-software.html. Luettu: 28.05.2020.

Suomen kiinteistölehti 2017. Puolet suomalaisista kokee asuvansa ahtaasti suuren tavaramäärän takia. Luettavissa: https://www.kiinteistolehti.fi/puolet-suomalaisista-kokee-asuvansa-ahtaasi-suuren-tavaramaaran-takia/. Luettu: 07.05.2020.

Telia 2016. Digitaaliset palvelut teollistuvat – Turha käsityö vähenee. Luettavissa: https://www.telia.fi/yrityksille/artikkelit/artikkeli/digitaaliset-palvelut-tehostuvat. Luettu: 12.05.2020.

Thinking Portfolio 25.07.2016. Kuinka valita sopiva menetelmä projektiin? Luettavissa: https://thinkingportfolio.com/kuinka-valita-sopiva-menetelma-projektiin/. Luettu: 14.05.2020.

TutorialsPoint s.a. What is XMLHttpRequest. Luettavissa: https://www.tutorialspoint.com/ajax/what_is_xmlhttprequest.htm. Luettu: 28.05.2020.

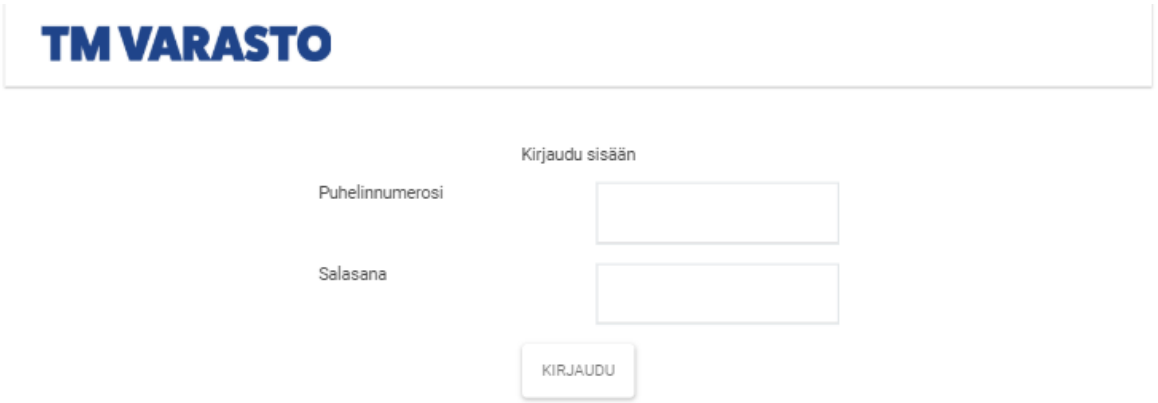
TypescriptLang 2020. What is TypeScript? Luettavissa: https://www.typescriptlang.org/. Luettu: 10.05.2020.

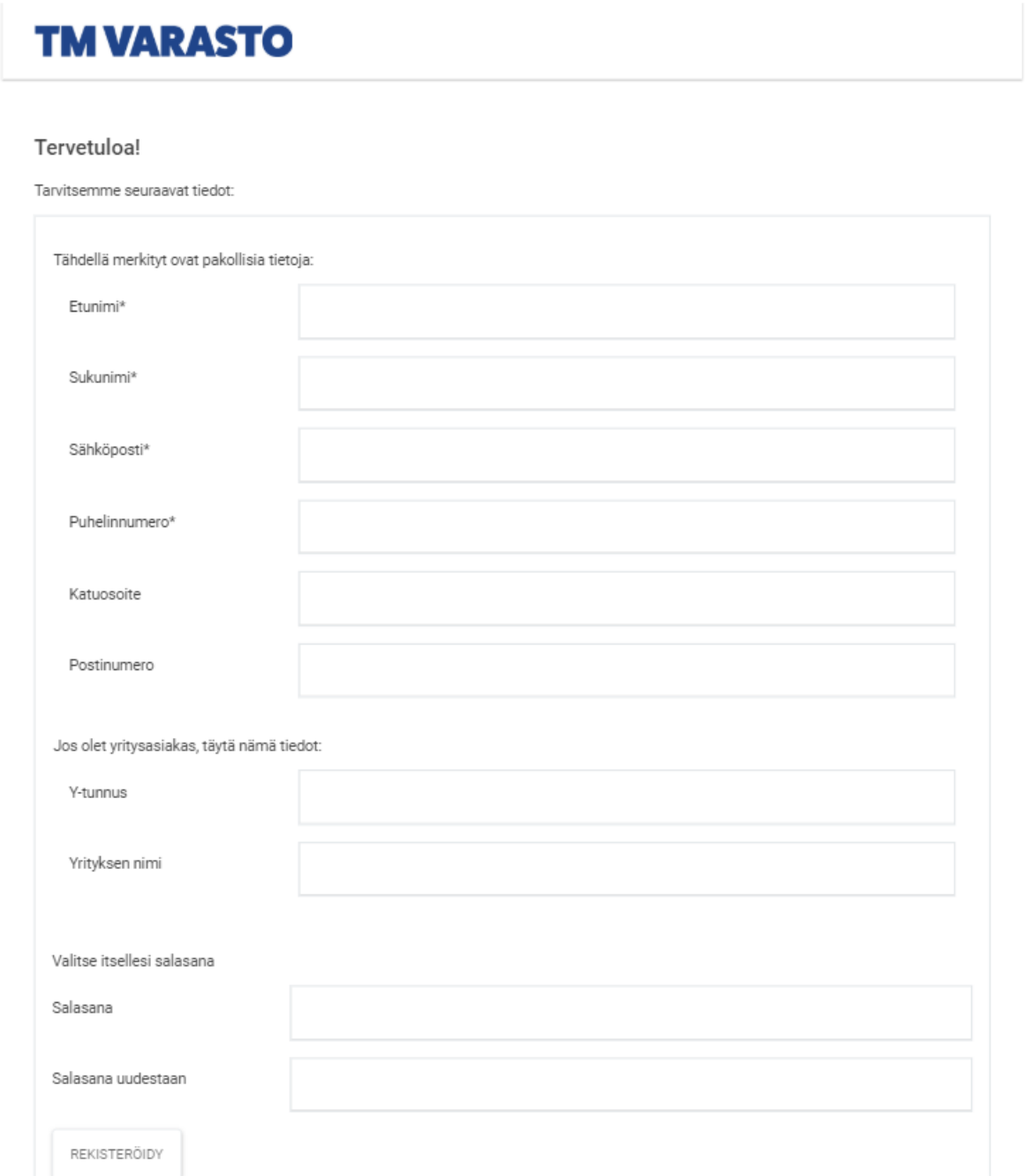
Valishvili L. 2018. Design with Precision – An Adobe XD Review. Luettavissa: https://www.toptal.com/designers/adobe/adobe-xd-review. Luettu: 10.05.2020.

Wpbeginner 23.08.2019. Beginner's Guide: What is a Domain Name and How Do Domains Work? Luettavissa: https://www.wpbeginner.com/beginners-guide/beginners-guide-what-is-a-domain-name-and-how-do-domains-work/. Luettu: 11.05.2020.

Liitteet

Liite 1. Sovelluksen kirjautumis- ja rekisteröitymisnäkymät





Liite 2. Sovelluksen palveluiden vuokrausnäkymä